Proceedings of ICAD2013
The Seventh International Conference on Axiomatic Design
Worcester – June 27-28, 2013
ICAD-2013-28

THE APPLICATION OF SEQUENCE ENUMERATION TO THE AXIOMATIC DESIGN PROCESS

Brandon Woolley
brandon.woolley@ttu.edu
Department of Mechanical Engineering
Texas Tech University
7th and Boston
Lubbock, TX 79409, USA

Zhen Li
zhen.li@ttu.edu
Department of Mechanical Engineering
Texas Tech University
7th and Boston
Lubbock, TX 79409, USA

Derrick Tate
d.tate@ttu.edu
Department of Mechanical Engineering
Texas Tech University
7th and Boston
Lubbock, TX 79409, USA

ABSTRACT

Today, software engineering is a well-defined structured discipline. Many new software engineers enter the workforce with a fundamental understanding of a software development life cycle. Unfortunately, new software engineers lack the necessary design techniques to move from requirements through the design phase. The idea of applying Axiomatic Design to software development was first proposed over two decades ago, yet is scarcely used today. Axiomatic Design provides a systematic approach to software design that programs of any size can use. This paper reviews several powerful attributes of Axiomatic Design for software engineering and evaluates the application of the embedded software engineering technique: sequence enumeration. In the case study, we show how to use both concepts seamlessly to yield a proper design for embedded systems.

Keywords: software engineering, Axiomatic Design, sequence enumeration.

1 INTRODUCTION

Traditionally, software programming was thought of as more art than science. Software engineering has evolved over the last forty years from simply programming or coding into the well-defined discipline that it is today. Through this evolution, software engineering has had countless software process models and various methodologies applied to it. These numerous process models were created to address the complexities associated with the software development life cycle. Each process model has advantages and disadvantages [Munassar and Govardhan, 2010]; however, all share one major disadvantage: They neglect the design phase. They also tend to over complicate the fundamental engineering process.

Axiomatic Design (AD) provides a basic established set of activities necessary for engineering design. Though it has been in use since the mid-nineties in other disciplines it hasn't garnered the similar attention from software engineering. Axiomatic Design facilitates the generation of only some the necessary software engineering artifacts for interphase transitions. Sequence enumeration can help fill in the artifact gap while providing a simple method for doing so.

Sequence enumeration is typically an embedded software engineering technique that provides the engineer with a formalized method for analyzing a system. It further aids the creation of a requirements specification that is in turn used to implement system state machine. Sequence enumeration is at the heart of creating a sequence-based software specification [Prowell, 1996]. Oshana [2006], used sequence enumeration as a method for developing use-case-based requirements specifications. This provides the embedded software engineer a valuable tool for creating correct end-to-end traceability in his or her designs.

2 BACKGROUND

2.1 RELATED WORK

Based on the work of Kim et al. [1991a; 1991b], Do and Suh extended the application of Axiomatic Design to software development to include object-oriented programming. Suh and Do illustrated the benefits of combining AD and object-oriented programming [Do and Suh, 2000; Do and Suh, 1999; Suh and Do, 2000]. These benefits include the ability to identify modules affected by a requirement change and a way to ensure low coupling through functional independence. AD also suggests the use of a design matrix to order the development tasks. For better understanding, consider Equation 1 [Suh, 2005].

$$\{FR\} = [A] * \{DP\} \tag{1}$$

The above relationship can be expanded to show the effect of the Independence Axiom on a design. For example, Figure 2 contains a functionally dependent (or coupled) design where more than one design parameter satisfies more than one functional requirement.

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & X & 0 \\ 0 & X & X \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} \qquad \begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix}$$

Figure 1. Coupled design (left) and decoupled design (right).

Additionally, the second part of Figure 1 represents a functionally decoupled (independent) design where the DPs and FRs have been rearranged into a lower-triangular matrix. This will provide the design enough functional independence by reducing the complexity (ergo coupling). The decoupled matrix also illustrates an order of task execution starting from the left side and moving to the right. This arrangement identifies the DPs with the most functional interdependence and these should be implemented first.

Pimentel and Stadzisz [2006] integrated AD with the unified software development process and utilized use cases to support functional decomposition. Moreover, they linked the need of a use-case-driven design to AD and functional requirement decomposition.

Schreyer and Tseng [2000], analyzed the application of Axiomatic Design to the design of PLC software. In their paper, Schreyer and Tseng illustrated the usefulness of state charts to support the decomposition and zigzagging of FRs and DPs. The key take-away was the application of state diagrams and sequence evaluation methods to the Axiomatic Design process.

Do [2004], pointed out that most software processes have difficulty dealing with changing requirements. As a result, most Unified Modeling Language (UML) tools intended to manage requirements are often used for tracking and reporting functions. This renders the tools irrelevant. Do goes on to demonstrate how the Axiomatic Design approach could benefit software product management.

2.2 SEQUENCE ENUMERATION

Sequence enumeration is an embedded software engineering technique used to expose buried requirements and for producing thorough specifications. The process ensures correct, complete, and traceable requirement specifications as well as a source for decisions. Oshana [2000], explained how this approach considered unforeseen permutations of stimuli to bring out ambiguities and omissions in the requirements. Prowell *et al.*, [1999], provided an orderly step-by-step process for defining system behavior and Oshana [2012], extended this into a systematic specification development method:

1. Establish the system boundary
2. Define the interfaces
3. Itemize the stimuli and the responses
4. Perform sequence enumeration
5. Identify the canonical sequence
6. Generate the state machine specification
7. Convert the state machine to code

Sequence enumeration is broadly applicable to many different types of systems. For example, it can be used to quickly model the behavior of a soda machine or to model the interfaces of a weapons system. The best way to express the usefulness of the sequence enumeration process is by example (see the next section).

3 CASE STUDY - SIMPLE WATCH EXAMPLE

Axiomatic Design has been used to augment the software engineering process to aid the design phase. Sequence enumeration can add more detail and fidelity in generating requirements as well as modelling initial system behavior. To illustrate the effectiveness of combining axiomatic design and sequence enumeration, a simple digital watch example is explored.

3.1 APPLYING AXIOMATIC DESIGN

The watch should display the time. A tick event should occur every second. And the time should be updated and output to display. In this paper, we concentrated on the watch's internal mechanism – tick and update. Therefore, two top FRs were:

FR1: Tick
FR2: Update watch

For a watch, buttons are often reused to perform multiple functions that are more practical for small devices such as a watch in our case. DP2 reflects this intuition. Equation 2 is the matrix for the top-level design.

DP1: Tick Event
DP2: Button sequential operations

$$\begin{bmatrix} FR1 \\ FR2 \end{bmatrix} = \begin{bmatrix} X & 0 \\ 0 & X \end{bmatrix} \begin{bmatrix} DP1 \\ DP2 \end{bmatrix} \quad (2)$$

Further decomposing FR2, we discovered several sub-FRs. And these FRs should be met with two buttons (Button A & Button B). The question is: how can we determine a sequence of buttons to satisfy five FRs? We rely on sequence enumeration to explore appropriate DPs.

FR2.1: Mode Change
FR2.2: Mode Change (hour)
FR2.3: Minute Set
FR2.4: Hour Set
FR2.5: Mode Update (normal)

3.2 APPLYING SEQUENCE ENUMERATION

In general, the fundamental progression for sequence enumeration is:

- Start with the smallest length stimulus sequences and define the appropriate response
- Record derived requirements as necessary
- Extend sequences that are not illegal or have equivalencies
- Continue until all sequences are either illegal or equivalent to previous sequences
- Identify the canonical sequences

Table 1. Simple watch requirements.

Req. #	Requirement
1	The watch displays the time and a tick event occurs every second; the time is updated and output to display
2	The watch has two external buttons A & B. Whenever 'A' is pressed in normal mode, the watch enters set mode, with minute update mode first
3	Each depression of 'B' causes the minutes field to update by 1(mod 60)
4	Pressing the 'A' button again will cause the watch to enter the hour update mode
5	Each successive depression of the 'B' button will increment the hour field by 1(mod 12)
6	Pressing 'A' again causes the watch to return to normal mode (displaying current time)

First, the requirements and DPs (buttons) are gathered in Table 1 using natural language in the voice of the customer.

With these requirements a system boundary definition with interfaces can be crafted. First, defining the system boundary allows for the identification of external interfaces. Generically speaking, the interfaces are the system's inputs and outputs. Once the interfaces are defined, the external stimuli and their corresponding responses can be drawn.

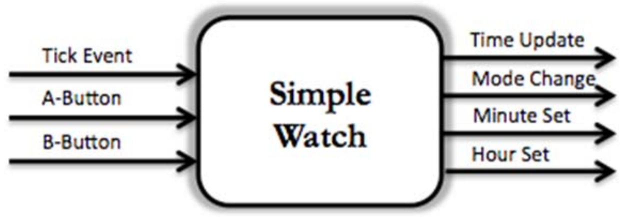

Figure 2. Simple watch system boundary.

Next, an itemized set of stimuli and responses (Table 2 and 3) can be created recording their requirements trace. Note the abstractions used are meant to obscure well-understood and previously recorded details. These abstractions are necessary for the management of the enumeration process.

Table 2. Itemized stimuli.

Stimuli	Description	Trace
Tick Event	Occurs every second	1
A-Button	Used to select time field to increment	2, 4, 6
B-Button	Used to increment the minute and hour fields	3, 5

Table 3. Itemized response.

Response	Description	Trace
Time Update	Updates the time accordingly	1
Mode Change	Cycles between minute, hour, and normal mode	2, 4, 6
Minute Set	Sets the minute field	3
Hour Set	Sets the hour field	5

There are two supplementary responses not identified in the system boundary nor the preceding itemizations:

- NULL Response – occurs when there is no external response for the given stimuli
- Illegal Response – is an impossible sequence

A stimulus can be illegal by definition or by design. An illegal by definition is one where it is impossible for the system to encounter it or for the system to generate it. An illegal by design is one that the system is designed explicitly to prevent. Moreover, a sequence can be 'equivalent' to another sequence if they share the responses to the same future stimuli. It is 'reduced' if it has been declared equivalent to a previous sequence. Finally, it is 'canonical' if it is legal and unreduced when the enumeration process is complete. The sequence enumeration process produces:

Table 4. Sequence enumeration.

Seq. #	Stimuli	Response	Equivalence	Req.
0	Empty	NULL		D1
1	T	Time Update		1
	A	Mode Change		2
	B	NULL	Empty	D2
2	TT	Time Update	T	1
	TA	Mode Change	A	2
	TB	NULL	B	D2
	AT	NULL	A	D3
	AA	Mode Change (hour)		4
	AB	Minute Set		3
3	AAT	NULL	AA	D3
	AAA	NULL	Empty	D3
	AAB	Hour Set		5
	ABT	NULL	AB	D3
	ABA	Mode Change (hour)	AA	4
	ABB	Minute Set	AB	3
4	AABT	NULL	AAB	D3
	AABA	Mode Update (normal)	Empty	6
	AABB	Hour Set	AAB	5

To reiterate, one of the most important aspects of sequence enumeration is that it can uncover unforeseen sequence permutations. These unforeseen permutations often become derived requirements. By definition, a derived requirement is one that is not defined by the customer but is generally uncovered by the design process. During the enumeration process it is normal to create, record, and include derived requirements like D1, D2, and D3. These newly added requirements become a part of the enumeration process and are evaluated accordingly. Notice that this simple system has equivalences at sequences of length 4 and the enumeration process is concluded. Each sequence has been mapped to a response providing a complete and consistent scenario of use. Enumeration exposes all possible, impossible, intended, and unintended uses of the system. A sequence of use characterizes a use case scenario.

The next step is canonical sequence analysis. This step is used to extract the sequences without equivalences, thereby constructing the canonical sequences depicted in Table 5:

Table 5. Canonical sequence.

Seq. #	Stimuli	Response	Equivalence	Req.
0	Empty	NULL		D1
1	T	Time Update		1
	A	Mode Change		2
2	AA	Mode Change (hour)		4
	AB	Minute Set		3
3	AAB	Hour Set		5
4	AABA	Mode Update (normal)		6

The canonical sequence table represents the legal and unique sequences for system usage. The analysis also reveals state data to be used to capture and preserve components of stimulus history to produce the correct system response. From the canonical sequence a state data table (Table 6) can be extracted. Also, we can use information from Table 5 to derive our DPs to meet sub-FRs derived from FR2.

DP2.1: A
DP2.2: A → A
DP2.3: A → B
DP2.4: A → A → B
DP2.5: A → A → B → A

The design matrix for FR2 can be re-written in the form of Equation 3. The matrix indicates that the design we obtained is a decoupled design. However, it's not likely to obtain an uncoupled form since the number of buttons is fewer than the number of FRs.

$$\begin{bmatrix} FR2.1 \\ FR2.2 \\ FR2.3 \\ FR2.4 \\ FR2.5 \end{bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 & 0 \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & X & 0 & X & 0 \\ X & X & X & X & X \end{bmatrix} \begin{bmatrix} DP2.1 \\ DP2.2 \\ DP2.3 \\ DP2.4 \\ DP2.5 \end{bmatrix} \quad (3)$$

Table 6. State data creation.

Sequence	**State Variable**	**Before Stimulus**	**After Stimulus**
Empty	N/A		
T; a tick event has occurred	MODE TIME	NORMAL CUR_TIME	NORMAL CUR_TIME+1 s
A; the user has pressed the A-button	MODE TIME	NORMAL CUR_TIME	SET_MIN CUR_TIME
AA; user pressed the A-button twice	MODE TIME	SET_MIN CUR_TIME	SET_HOUR CUR_TIME
AB; user pressed the A then B-button	MODE TIME	SET_MIN CUR_TIME	SET_MIN CUR_TIME+1 m
AAB; user pressed the A-button twice followed by the B-button	MODE TIME	SET_HOUR CUR_TIME	SET_HOUR CUR_TIME+1 h

The newly created variables represent state data for the system. These state variables can then be recast into a state-based specification using natural language. Generation of the following state transition diagram in Figure 5 is the last artifact necessary before implementation.

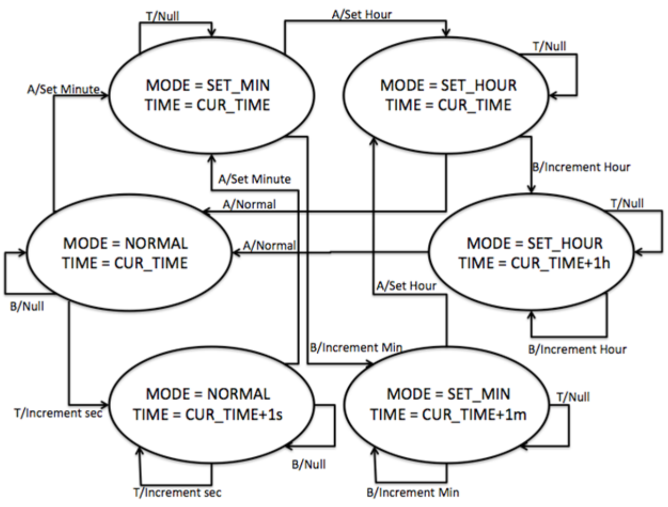

Figure 3. Simple watch state transition diagram.

It should also be noted that the sequence enumeration process calls for the conversion of the state transition diagram to source code. A step that can use the information in Table 6 can be automated.

As indicated by Oshana [2006], sequence enumeration provides complete, consistent, traceable, verifiably correct specifications. For example, each element of the state-based specification can be compared to the sequence-based specification to confirm that correctness is preserved.

4 DISCUSSION

Axiomatic Design and sequence enumeration are employed to deal with complexity within their respective disciplines. Sequence enumeration and Axiomatic Design have a set of complementary design activities. AD is used at a higher level while sequence enumeration is generally employed at a lower level. The deployment of sequence enumeration helps explore proper DPs at low-level design without sacrificing exhaustiveness of all logical sequences. The design matrix derived from sequence enumeration can be used for determining if the Independence Axiom is satisfied or not. The integration of two theories makes it possible to yield a design with the low complexity (avoid coupled design) and high completeness (ensured by sequence enumeration).

There have been other approaches to enhance Axiomatic Design for software. Do's early work [Do and Suh, 1999] highlighted the application of AD to OOP to ensure a higher degree of functional independence while Schreyer and Tseng [2000], applied state charts to support decomposition, and Pimentel and Stadzisz [2006], employed use case based OO software design.

The sequence enumeration process has many practical advantages for software engineering. The process provides various artifacts for specifications and provides a systematic method for development. Combining both AD and sequence enumeration has the potential to enhance the software design phase by adding greater detail and fidelity. Furthermore, sequence enumeration aids the generation of a system model that early AD phases will benefit from. The advantages of sequence enumeration emphasized by Oshana, in [2000] are:

- The ability to model system functionality early
- Provide the customer operational system understanding
- A conduit to analyse and improve functional requirements

The lower level applicability of sequence enumeration can augment the AD process to provide some measure of checks and balances. Further investigation will be needed in order to develop a more formalized model or framework of integration.

5 REFERENCES

[1] Do, S.-H., "Software product lifecycle management using axiomatic approach". *The 3rd International Conference on Axiomatic Design*, 2004.

[2] Do, S.-H. and Suh, N. P., "Systematic OO programming with axiomatic design." *Computer*, Vol. 32, No. 10, pp.121-124, 1999.

[3] Do, S.-H. and Suh, N. P., "Object-oriented software design with axiomatic design". *Proceedings of the 1st International Conference on Axiomatic Design*, Cambridge, MA, 2000.

[4] Kim, S.-J., Suh, N. and Kim, S.-G., "Design of software system based on axiomatic design." *CIRP Annals-Manufacturing Technology*, Vol. 40, No. 1, pp.165-170, 1991.

[5] Kim, S.-J., Suh, N. P. and Kim, S.-G., "Design of software systems based on axiomatic design." *Robotics and Computer-Integrated Manufacturing*, Vol. 8, No. 4, pp.243-255, 1991.

[6] Munassar, N. M. A. and Govardhan, A., "A Comparison Between Five Models Of Software Engineering." *IJCSI International Journal of Computer Science Issues*, Vol. 7, No. 5, pp.94-101, 2010.

[7] Oshana, R., "Sequence Enumeration." from http://www.embedded.com/design/debug-and-optimization/4403169/Sequence-Enumeration, 2000.

[8] Oshana, R., *DSP software development techniques for embedded and real-time systems*, Newnes, 2006.

[9] Oshana, R., *DSP for Embedded and Real-time Systems*, Newnes, 2012.

[10] Pimentel, A. R. and Stadzisz, P. C., "A use case based object-oriented software design approach using the axiomatic design theory". *The 4th International Conference on Axiomatic Design*, 2006.

[11] Prowell, S. J., "Sequence-based software specification", University of Tennessee, Knoxville, 1996.

[12] Prowell, S. J., Trammell, C. J., Linger, R. C. and Poore, J. H., *Cleanroom software engineering: technology and process*, Addison-Wesley Professional, 1999.

[13] Schreyer, M. and Tseng, M., "Hierarchical State Decomposition for Design of PLC Software by applying Axiomatic Design". *Proceedings of the 1st International Conference on Axiomatic Design*, Cambridge, MA, 2000.

[14] Suh, N. P., *Complexity: Theory and Applications*, Oxford University Press on Demand, 2005.

[15] Suh, N. P. and Do, S.-H., "Axiomatic design of software systems." *CIRP Annals-Manufacturing Technology*, Vol. 49, No. 1, pp.95-100, 2000.